**高中物理引擎**

**引言：**

相信许多学生都经受过高中物理计算大题的折磨。一方面计算大题的思路比较复杂运动过程难以直白的在脑中刻画；另一方面计算大题的计算量十分巨大。

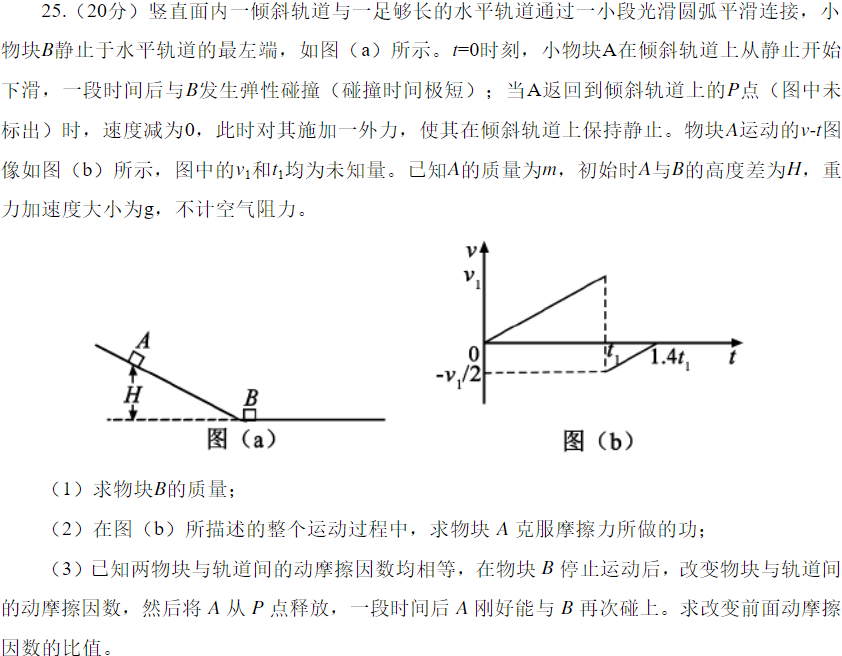
不得不说计算大题确实较好地贯彻了物理学的精髓。从百度百科中搜索到对物理学的定义可以看出物理学的两个特点：1. 研究物质运动最一般规律 2. 运用数学作为工作语言，这分别对应了计算大题的两个难点。

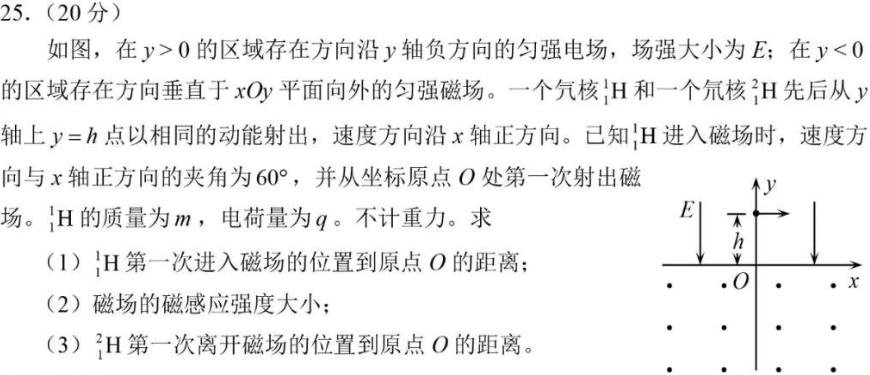
但是对于基础较差尤其是想象力较弱，计算能力有待提高的学生而言，计算大题确实是不小的挑战。因此，我想通过一些简单的方式对高中物理计算大题的运动过程进行模拟。一方面可以帮助基础较差的学生更加直观的了解到运动过程，另一方面可以通过查看数据进行答案的比对和对具体哪一步计算有误进行快速查找。

**功能构思：**

首先希望它可以直观地显示物理模型的运动过程，其次希望可以从中获取一些物理量具体运算结果。

**开发计划：**





通过对几年的物理大题的总结，可以看出物理大题中的一些元素：小球/小木块，斜坡，杆/绳，电/磁场……需要我们计算的物理量无非就是：受力，位置，做功……

首先根据力的定义：力是物体对物体的作用，力不能脱离物体而单独存在；和在物理计算中总结的经验：计算受力一般就是计算运动的加速度即力更像是一个人为规定的用于描述两物体相互作用时其运动和质量的关系的概念。因此，我在此将力作为一个独立于运动的单独的概念：关系。

那么按道理说，场作为描述力和电荷，质量等另一个物理量的关系的对象，应该也归到和力同样的类中。但是可以看出计算大题计算中场力和真实的力的一个差别：物理计算大题中的场都是不管施力物体的，比如电场，我们不考虑发出电场的物体的运动如何。因此将电场磁场重力场作为独立于关系的一个单独的概念：场。

然后就是小球，小木块等等，这些在物理计算大题中，基本上都是有一些属性（质量，电荷量），需要计算其运动状态的一个物体。完全不同于关系和场，因此将其作为一个独立于关系和场的概念：物体。

然后对剩下的元素进行分类。斜坡，重力都是属于不需要关注施力物体的，并会对物体运动产生影响的因而将其归到场。杆，绳有的时候是可以连接两个物体并分别影响两物体的运动的，类似于力的概念，因而将其归为关系。位置，速度这些运动状态是需要计算的，因而将其归为物体的属性。做功则是力对运动距离的积分，可以用物体运动质量等属性进行进一步计算，因而不特别的归为一类。

综上所述，通过这种分类：物体——计算的对象、属性的集合，关系——两个物体属性的相互影响，场——对单个物体属性的影响，可以对高中物理计算大题进行一个模拟。

以下是此部分的代码实现：

import sys

class Relation:

def \_\_init\_\_(self, obj1, obj2):

self.\_\_link = tuple(obj1, obj2)

class LightString(Relation):

def \_\_init\_\_(self, obj1, obj2, length, k=sys.float\_info.max, C=sys.float\_info.max):

super().\_\_init\_\_(obj1, obj2)

self.\_\_length = length

self.\_\_k = k

self.\_\_C = C

def calculate(self):

delta\_x = self.\_\_link[1].pos[0] - self.\_\_link[0].pos[0]

delta\_y = self.\_\_link[1].pos[1] - self.\_\_link[0].pos[1]

distance = (delta\_x\*\*2 + delta\_y\*\*2)\*\*0.5

if distance >= self.length:

cos = delta\_x / distance

sin = delta\_y / distance

Fr = (self.\_\_link[1].v[0] \* cos + self.\_\_link[1].v[1] \* sin

- (self.\_\_link[0].v[0] \* cos + self.\_\_link[0].v[1] \* sin)) \* self.\_\_C

F\_x = self.\_\_k \* (delta\_x - self.\_\_length \* cos) + Fr \* cos

F\_y = self.\_\_k \* (delta\_y - self.\_\_length \* sin) + Fr \* sin

self.\_\_link[0].a[0] += F\_x / self.\_\_link[0].m

self.\_\_link[0].a[1] += F\_y / self.\_\_link[0].m

self.\_\_link[1].a[0] -= F\_x / self.\_\_link[1].m

self.\_\_link[1].a[1] -= F\_y / self.\_\_link[1].m

self.draw()

def draw(self):

pass

class LightSpring(Relation):

def \_\_init\_\_(self, obj1, obj2, length, k, C=0):

super().\_\_init\_\_(obj1, obj2)

self.\_\_length = length

self.\_\_k = k

def calculate(self):

delta\_x = self.\_\_link[1].pos[0] - self.\_\_link[0].pos[0]

delta\_y = self.\_\_link[1].pos[1] - self.\_\_link[0].pos[1]

distance = (delta\_x\*\*2 + delta\_y\*\*2)\*\*0.5

cos = delta\_x / distance

sin = delta\_y / distance

Fr = (self.\_\_link[1].v[0] \* cos + self.\_\_link[1].v[1] \* sin

- (self.\_\_link[0].v[0] \* cos + self.\_\_link[0].v[1] \* sin)) \* self.\_\_C

F\_x = self.\_\_k \* (delta\_x - self.\_\_length \* cos) + Fr \* cos

F\_y = self.\_\_k \* (delta\_y - self.\_\_length \* sin) + Fr \* sin

self.\_\_link[0].a[0] += F\_x / self.\_\_link[0].m

self.\_\_link[0].a[1] += F\_y / self.\_\_link[0].m

self.\_\_link[1].a[0] -= F\_x / self.\_\_link[1].m

self.\_\_link[1].a[1] -= F\_y / self.\_\_link[1].m

self.draw()

def draw(self):

pass

class LightStick(LightSpring):

def \_\_init\_\_(self, obj1, obj2, length):

super().\_\_init\_\_(obj1, obj2, length, sys.float\_info.max, sys.float\_info.max)

class Gravitation(Relation):

def \_\_init\_\_(self, obj1, obj2, G=6.67428\*10\*\*(-11)):

super().\_\_init\_\_(obj1, obj2)

self.\_\_G = G

def calculate(self):

delta\_x = self.\_\_link[1].pos[0] - self.\_\_link[0].pos[0]

delta\_y = self.\_\_link[1].pos[1] - self.\_\_link[0].pos[1]

distance = (delta\_x\*\*2 + delta\_y\*\*2)\*\*0.5

cos = delta\_x / distance

sin = delta\_y / distance

F = self.\_\_G \* self.\_\_link[0].m \* self.\_\_link[1].m / distance\*\*2

F\_x = F \* cos

F\_y = F \* sin

self.\_\_link[0].a[0] += F\_x / self.\_\_link[0].m

self.\_\_link[0].a[1] += F\_y / self.\_\_link[0].m

self.\_\_link[1].a[0] -= F\_x / self.\_\_link[1].m

self.\_\_link[1].a[1] -= F\_y / self.\_\_link[1].m

class ElectrostaticForce(Relation):

def \_\_init\_\_(self, obj1, obj2, k=8.987551\*10\*\*9):

super().\_\_init\_\_(obj1, obj2)

self.\_\_k = k

def calculate(self):

delta\_x = self.\_\_link[1].pos[0] - self.\_\_link[0].pos[0]

delta\_y = self.\_\_link[1].pos[1] - self.\_\_link[0].pos[1]

distance = (delta\_x\*\*2 + delta\_y\*\*2)\*\*0.5

cos = delta\_x / distance

sin = delta\_y / distance

F = self.\_\_k \* self.\_\_link[0].q \* self.\_\_link[1].q / distance\*\*2

F\_x = F \* cos

F\_y = F \* sin

self.\_\_link[0].a[0] += F\_x / self.\_\_link[0].m

self.\_\_link[0].a[1] += F\_y / self.\_\_link[0].m

self.\_\_link[1].a[0] -= F\_x / self.\_\_link[1].m

self.\_\_link[1].a[1] -= F\_y / self.\_\_link[1].m

显示方面则是用pygame实现，在此不做过多展示。

**总结：**

当前存在一定问题，如通过多个弹簧模拟的绳子会不断变长。分析原因如下：1. 单位问题，目前相当于将单位设为1px对应1m 2. 积分计算不精确，目前是用达布下和近似代替积分。

**参考资料：**

<https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%AD%A6/313183?fr=aladdin>

<https://wenku.baidu.com/view/90c11aa5970590c69ec3d5bbfd0a79563d1ed454.html>

<https://wenku.baidu.com/view/d8a577dbdc3383c4bb4cf7ec4afe04a1b171b06c.html>

<https://blog.csdn.net/danieldingshengli/article/details/82226345>

<https://www.jianshu.com/p/a425d2bb32c1>

http://buildnewgames.com/gamephysics/